ARM® Cortex™-A9 processors

r4 releases

Software Developers Errata Notice

ARM UAN 0009B (ID102312)

ARM Cortex-A9 processors
Software Developers Errata Notice

Release Information

The following changes have been made to this book.

Change History

Date	Issue	Confidentiality	Change
18 June 2012	A	Non-confidential	First release for r4 releases, limited distribution
20 September 2012	B	Non-confidential	Second release for r4 releases

Proprietary Notice

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

Contents
ARM Cortex-A9 processors Software Developers Errata Notice

Chapter 1
Introduction

This chapter introduces the errata notices for ARM Cortex™-A9 processors, for r4 releases.

1.1 Scope of this document

This document describes errata categorized by level of severity. Each description includes:

- The current status of the defect.
- Where the implementation deviates from the specification and the conditions under which erroneous behavior occurs.
- The implications of the erratum with respect to typical applications.
- The application and limitations of a 'work-around' where possible.

This document describes errata that may impact anyone who is developing software that will run on implementations of this ARM product.

1.2 Categorization of errata

Errata recorded in this document are split into the following levels of severity:

Table 1-1 Categorization of errata

Errata type	Definition
Category A	A critical error. No workaround is available or workarounds are impactful. The error is likely to be common for many systems and applications.
Category A (rare)	A critical error. No workaround is available or workarounds are impactful. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category B	A significant error, or a critical error with an acceptable workaround. The error is likely to be common for many systems and applications.
Category B (rare)	A significant error, or a critical error with an acceptable workaround. The error is likely to be rare for most systems and applications. Rare is determined by analysis, verification and usage.
Category C	A minor error.

1.3 Errata summary

Table 1-2 lists all the errata described in this document. The Status column shows any errata that are new or updated in the current issue of the document. An erratum is shown as updated if there has been any change to the text of the erratum Description, Conditions, Implications or Workaround. Fixed errata are not shown as updated, unless the erratum text has changed.

Table 1-2 List of errata

Status	ID	Area	Cat	Rare	Summary of erratum
-	761319	Prog	A	Rare	Possible inconsistent sequencing of read accesses to the same memory location
-	740657	Prog	B	-	Global Timer can send two interrupts for the same event
-	751476	Prog	B	-	Missed watchpoint on the second part of an unaligned access crossing a page boundary
-	754322	Prog	B	-	Faulty MMU translations following ASID switch
Updated	764369	Prog	B	-	Data or unified cache line maintenance by MVA fails on Inner Shareable memory
New	794072	Prog	B	-	A short loop including a DMB instruction might cause a denial of service on another processor which executes a CP15 broadcast operation
New	794073	Prog	B	-	Speculative instruction fetches with MMU disabled might not comply with architectural requirements
New	794074	Prog	B	-	A write request to Uncacheable, Shareable normal memory region might be executed twice, possibly causing a software synchronisation issue
-	725631	Prog	C	-	ISB is counted in Performance Monitor events 0x0C and 0x0D
-	729817	Prog	C	-	Main ID register alias addresses are not mapped on Debug APB interface
-	729818	Prog	C	-	In debug state, next instruction is stalled when SDABORT flag is set, instead of being discarded
-	751471	Prog	C	-	DBGPCSR format is incorrect
-	752519	Prog	C	-	An imprecise abort might be reported twice on non-cacheable reads
-	754323	Prog	C	-	Repeated Store in the same cache line might delay the visibility of the Store
-	756421	Prog	C	-	Sticky Pipeline Advance bit cannot be cleared from debug APB accesses
-	757119	Prog	C	-	Some *Unallocated memory hint* instructions generate an Undefined Instruction exception instead of being treated as NOP
-	761321	Prog	C	-	MRC and MCR are not counted in event 0x68

Table 1-2 List of errata (continued)

Status	ID	Area	Cat	Rare	Summary of erratum
-	764319	Prog	C	-	Read accesses to DBGPRSR and DBGOSLSR may generate an unexpected Undefined Instruction exception
-	771224	Prog	C	-	Visibility of Debug Enable access rights to enable/disable tracing is not ensured by an ISB
-	775419	Prog	C	-	PMU event 0x0A (exception return) might count twice the LDM PC ^ instructions with base address register write-back

Chapter 2
Errata Descriptions

This chapter includes the errata description for ARM Cortex™-A9 processors, for r4 releases.

2.1 Category A

There are no errata in this category.

2.2 Category A (Rare)

This section describes Category A rare errata.

2.2.1 (761319) Ordering of read accesses to the same memory location might be uncertain

Status

Affects: Product Cortex-A9 MPCore

Fault Type: Programmer Category A (Rare)

Fault Status: Present in: All revisions. Open

Description

The ARM architecture and the general rules of coherency require reads to the same memory location to be observed in sequential order.

Because of some internal replay path mechanisms, the Cortex-A9 can see one read access bypassed by a following read access to the same memory location, thus not observing the values in program order.

Conditions

The erratum requires a Cortex-A9 MPCore configuration with two or more processors or more.

The erratum can occur only on a processor working in SMP mode, on memory regions marked as Normal Memory Write-Back Shared.

Implications

The erratum causes data coherency failure.

Workaround

The majority of multi-processing code examples follow styles that do not expose the erratum. Therefore, this erratum occurs rarely and is likely to affect only very specific areas of code that rely on a read-ordering behavior.

There are two possible workarounds for this erratum:

- The first possible workaround is to use `LDREX` instead of standard `LDR` in volatile memory places that require a strict read ordering.
- The alternative possible workaround is the recommended workaround for tool chains integration. It requires insertion of a `DMB` between the affected `LDR` that requires this strict ordering rule.

For more information about integrating the workaround inside tool chains, see the Programmer Advice Notice related to this erratum, *ARM UAN 0004A*.

2.3 Category B

This section describes Category B errata.

2.3.1 (740657) Global Timer can send two interrupts for the same event

Status

Affects: Product Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r2, r3 and r4 revisions. Open

Description

The Global Timer can be programmed to generate an interrupt request to the processor when it reaches a given programmed value. Because of the erratum, when you program the Global Timer to not use the auto-increment feature, it might generate two interrupt requests instead of one.

Conditions

The Global Timer Control register is programmed with the following settings:

- Bit[3] = 1'b0 - Global Timer is programmed in single-shot mode
- Bit[2] = 1'b1 - Global Timer IRQ generation is enabled
- Bit[1] = 1'b1 - Global Timer value comparison with Comparator registers is enabled
- Bit[0] = 1'b1 - Global Timer count is enabled.

With these settings, an IRQ is generated to the processor when the Global Timer value reaches the value programmed in the Comparator registers. The Interrupt Handler then performs the following sequence:

- Read the ICCIAR (Interrupt Acknowledge) register
- Clear the Global Timer flag
- Modify the comparator value, to set it to a higher value
- Write the ICCEOIR (End of Interrupt) register.

Under these conditions, because of the erratum the Global Timer might generate a second (spurious) interrupt request to the processor at the end of this Interrupt Handler sequence.

Implications

The erratum creates spurious interrupt requests in the system.

Workaround

Because the erratum happens only when the Global Timer is programmed in single-shot mode, that is, when it does not use the auto-increment feature, a first possible workaround is to program the Global Timer to use the auto-increment feature.

If this first solution is not possible, a second workaround is to modify the Interrupt Handler to avoid the offending sequence. You can achieve this by clearing the Global Timer flag after incrementing the Comparator register value. The correct code sequence for the Interrupt Handler should then look like the following sequence:

- Read the ICCIAR (Interrupt Acknowledge) register

- Modify the comparator value, to set it to a higher value
- Clear the Global Timer flag
- Clear the Pending Status information for Interrupt 27 (Global Timer interrupt) in the Distributor of the Interrupt Controller.
- Write the ICCEOIR (End of Interrupt) register.

2.3.2 (751476) Missed watchpoint on the second part of an unaligned access crossing a page boundary

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore

Fault Type: Programmer Category B

Fault Status: Present in: All revisions. Open

Description

Under rare conditions, a watchpoint might be undetected if it occurs on the second part of an unaligned access that crosses a 4K page boundary and misses in the µTLB for the second part of its request.

The erratum requires a previous conditional instruction which accesses the second 4KB memory region (=where the watchpoint is set), which misses in the µTLB, and which is condition failed. The erratum also requires that no other µTLB miss occurs between this conditional failed instruction and the unaligned access, which implies that the unaligned access must hit in the µTLB for the first part of its access

Implications

A watchpoint does not trigger when it should.

Workaround

The erratum might occur in the case when a watchpoint is set on any of the first 3 bytes of a 4KB memory region, and unaligned accesses are not being faulted.

The workaround is then to set a guard watchpoint on the last byte of the previous page, and to deal with any false positive matches if they occur.

2.3.3 (754322) Faulty MMU translations following ASID switch

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r2, r3 and r4 revisions. Open

Description

A µTLB entry might be corrupted following an ASID switch, possibly corrupting subsequent MMU translations.

The erratum requires execution of a speculative explicit memory access, which speculation fails. The speculation fails if the memory access occurs under a mispredicted branch, or if it is conditional and the condition fails.

This speculative memory access might miss in the TLB, and cause a page table walk. The erratum occurs when the page table walk starts before the ASID switch code sequence, but completes after the ASID switch code sequence.

In this case, the µTLB gets a new entry allocated with this new TLB entry, corresponding to the old ASID. The erratum arises from failure of the µTLB to register the ASID value, so that MMU translations, which should happen with the new ASID following the ASID switch, might hit in this stale µTLB entry and become corrupted.

Note that there is no Trustzone Security risk because the Security state of the access is registered in the µTLB, and cannot be corrupted.

Implications

The erratum might cause MMU translation corruptions.

Workaround

The workaround for this erratum is to add a DSB in the ASID switch code sequence.

The ARM architecture only mandates ISB before and after the ASID switch. Adding a DSB before the ASID switch ensures that the page table walk completes before the ASID change, so that no stale entry can be allocated in the µTLB.

Modify the examples in the *ARM Architecture Reference Manual* for synchronizing the change in the ASID and TTBR as follows:

1. The sequence:

```
Change ASID to 0
ISB
Change Translation Table Base Register
ISB
Change ASID to new value
```

   Becomes:

```
DSB
Change ASID to 0
ISB
Change Translation Table Base Register
ISB
DSB
Change ASID to new value
```

2. The sequence:

```
Change Translation Table Base Register to the global-only mappings
ISB
Change ASID to new value
ISB
```

```
Change Translation Table Base Register to new value
```

Becomes:

```
Change Translation Table Base Register to the global-only mappings
ISB
DSB
Change ASID to new value
ISB
Change Translation Table Base Register to new value
```

3. And the sequence:

```
Set TTBCR.PD0 = 1
ISB
Change ASID to new value
Change Translation Table Base Register to new value
ISB
Set TTBCR.PD0 = 0
```

Becomes:

```
Set TTBCR.PD0 = 1
ISB
DSB
Change ASID to new value
Change Translation Table Base Register to new value
ISB
Set TTBCR.PD0 = 0
```

2.3.4 (764369) Data or unified cache line maintenance by MVA fails on Inner Shareable memory

Status

Affects: Product Cortex-A9 MPCore

Fault Type: Programmer Category B

Fault Status: Present in: All revisions. Open

Description

Under certain timing circumstances, a data or unified cache line maintenance operation by MVA that targets an Inner Shareable memory region might fail to propagate to either the Point of Coherency or to the Point of Unification of the system.

As a consequence, the visibility of the updated data might not be guaranteed to either the instruction side, in the case of self-modifying code, or to an external non-coherent agent, such as a DMA engine.

Conditions

The erratum requires a Cortex-A9 MPCore configuration with two or more processors, working in SMP mode, with the broadcasting of CP15 maintenance operations enabled.

The following scenario shows how the erratum can occur:

1. One CPU performs a data or unified cache line maintenance operation by MVA targeting a memory region which is locally dirty.
2. A second CPU issues a memory request targeting this same memory location within the same time frame.

A race condition can occur, resulting in the cache operation not being performed to the specified Point of Unification or Point of Coherence.

The erratum affects the following maintenance operations:

- DCIMVAC: Invalidate data or unified cache line by MVA to PoC
- DCCMVAC: Clean data or unified cache line by MVA to PoC
- DCCMVAU: Clean data or unified cache line by MVA to PoU
- DCCIMVAC: Clean and invalidate data or unified cache line by MVA to PoC.

The erratum can occur when the second CPU performs any of the following operations:

- A read request resulting from any Load instruction; the Load might be a speculative one
- A write request resulting from any Store instruction
- A data prefetch resulting from a PLD instruction; the PLD might be a speculative one.

Implications

Because it is uncertain whether execution of the cache maintenance operation propagates to either the Point of Unification or the Point of Coherence, stale data might remain in the data cache and not become visible to other agents that should have gained visibility on it.

Note that the data remains coherent on the L1 Data side. Any data read from another processor in the Cortex A9 MPCore cluster, or from the ACP, would see the correct data. In the same way, any write on the same cache line from another processor in the Cortex-A9 MPCore cluster, or from the ACP, does not cause a data corruption resulting from a loss of either data.

Consequently, the failure can only impact non-coherent agents in the systems. This can be either the instruction cache of the processor, in the case of self-modifying code, or any non-coherent external agent in the system like a DMA.

Workaround

Two workarounds are available for this erratum.

The first workaround requires the three following elements to be applied altogether:

1. Set bit[0] in the undocumented SCU Diagnostic Control register located at offset 0x30 from the PERIPHBASE address.

   Setting this bit disables the *migratory bit* feature. This forces a dirty cache line to be evicted to the lower memory subsystem, which is both the Point of Coherency and the Point of Unification, when it is being read by another processor.

2. Insert a DSB instruction before the cache maintenance operation.

   Note that, if the cache maintenance operation executes within a loop that performs no other memory operations, ARM recommends only adding a DSB before entering the loop.

3. Ensure there is no false sharing (on a cache line size alignment) for self-modifying code or for data produced for external non-coherent agent such as a DMA engine.

   For systems which cannot prevent false sharing in these regions, this third step can be replaced by performing the sequence of DSB followed by Cache maintenance operation twice.

Note that even when all three components of the workaround are in place, the erratum might still occur. However, this would require some extremely rare and complex timing conditions, so that the probability of reaching the point of failure is extremely low. This, and the fact that the erratum requires an uncommon software scenario, explains why this workaround is likely to be a reliable practical solution for most systems.

To ARM's knowledge, no failure has been observed in any system when all three components of this workaround have been implemented.

For critical systems that cannot cope with the extremely low failure risks associated with the above workaround, a second workaround is possible which involves changing the mapping of the data being accessed so that it is in a Non-Cacheable area. This ensures that the written data remains uncached. This means it is always visible to non-coherent agents in the system, or to the instruction side in the case of self-modifying code, without any need for cache maintenance operation.

2.3.5 (794072) A short loop including a DMB instruction might cause a denial of service on another processor which executes a CP15 broadcast operation

Status

Affects: Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All r1, r2, r3 and r4 revisions. Open

Description

A processor which continuously executes a short loop containing a DMB instruction might prevent a CP15 operation broadcast by another processor making further progress, causing a denial of service.

Configurations affected

This erratum affects all Cortex-A9 MPCore processors with two or more processors.

Conditions

The erratum requires the following conditions:

- Two or more processors are working in SMP mode (ACTLR.SMP=1)
- One of the processors continuously executes a short loop containing at least one DMB instruction.
- Another processor executes a CP15 maintenance operation that is broadcast. This requires that this processor has enabled the broadcasting of CP15 operations (ACTLR.FW=1)

For the erratum to occur, the short loop containing the DMB instruction must meet all of the following additional conditions:

- No more than 10 instructions other than the DMB are executed between each DMB
- No non-conditional Load or Store, or conditional Load or Store which pass the condition code check, are executed between each DMB

When all the conditions for the erratum are met, the short loop is creating a continuous stream of DMB instructions. This might cause a denial of service, by preventing the processor executing the short loop from executing the received broadcast CP15 operation. As a result, the processor that originally executed the broadcast CP15 operation is stalled until the execution of the loop is interrupted.

Note that because the process issuing the CP15 broadcast operation cannot complete operation, it cannot enter any debug-mode, and cannot take any interrupt. If the processor executing the short loop also cannot be interrupted, for example if it has disabled its interrupts, or if no interrupts are routed to this processor, this erratum might cause a system livelock.

Implications

The erratum might create performance issues, or in the worst case it might cause a system livelock if the processor executing the DMB is in an infinite loop that cannot be interrupted.

Workaround

This erratum can be worked round by setting bit[4] of the undocumented Diagnostic Control Register to 1. This register is encoded as CP15 c15 0 c0 1.

This bit can be written in Secure state only, with the following Read/Modify/Write code sequence:

```
MRC p15,0,rt,c15,c0,1
```

```
ORR rt,rt,#0x10
MCR p15,0,rt,c15,c0,1
```

When it is set, this bit causes the DMB instruction to be decoded and executed like a DSB.

Using this software workaround is not expected to have any impact on the overall performance of the processor on a typical code base.

Other workarounds are also available for this erratum, to either prevent or interrupt the continuous stream of DMB instructions that causes the deadlock. For example:

- Inserting a non-conditional Load or Store instruction in the loop between each DMB
- Inserting additional instructions in the loop, such as NOPs, to avoid the processor seeing back to back DMB instructions.
- Making the processor executing the short loop take regular interrupts.

2.3.6 (794073) Speculative instruction fetches with MMU disabled might not comply with architectural requirements

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All revisions. Open

Description

When the MMU is disabled, an ARMv7 processor must follow some architectural rules regarding speculative fetches and the addresses to which these can be initiated. These rules avoid potential read accesses to read-sensitive areas. For more information about these rules see the description of "Behavior of instruction fetches when all associated MMUs are disabled" in the *ARM Architecture Reference Manual*, ARMv7-A and ARMv7-R edition.

A Cortex-A9 processor usually operates with both the MMU and branch prediction enabled. If the processor operates in this condition for any significant amount of time, the *Branch Target Address Cache* (BTAC) will contain branch predictions. If the MMU is then disabled, but branch prediction remains enabled, these stale BTAC entries can cause the processor to violate the rules for speculative fetches.

Configurations affected

This erratum affects all configurations of the processor.

Conditions

The erratum can occur only if the following sequence of conditions is met:

1. MMU and branch prediction are enabled.
2. Branches are executed.
3. MMU is disabled, and branch prediction remains enabled.

Implications

If the above conditions occur, it is possible that after the MMU is disabled, speculative instruction fetches might occur to read-sensitive locations.

Workaround

The recommended workaround is to invalidate all entries in the BTAC, by executing an *Invalidate Entire Branch Prediction Array* (BPIALL) operation followed by a DSB, before disabling the MMU.

Another possible workaround is to disable branch prediction when disabling the MMU, and keep branch prediction disabled until the MMU is re-enabled.

2.3.7 (794074) A write request to Uncacheable, Shareable normal memory region might be executed twice, possibly causing a software synchronisation issue

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category B

Fault Status: Present in: All revisions. Open

Description

Under certain timing circumstances specific to the Cortex-A9 microarchitecture, a write request to an Uncacheable, Shareable Normal memory region might be executed twice, causing the write request to be sent twice on the AXI bus. This might happen when the write request is followed by another write into the same naturally aligned doubleword memory region, without a DMB between the two writes.

The repetition of the write usually has no impact on the overall behaviour of the system, unless the repeated write is used for synchronisation purposes.

Configurations affected

The erratum affects all configurations of the processor.

Conditions

The erratum requires the following conditions:

1. A write request is performed to an Uncacheable, Shareable Normal memory region.
2. Another write request is performed into the same naturally doubleword aligned memory region. This second write request must not be performed to the exact same bytes as the first store.

A write request to Normal memory region is treated as Uncacheable in the following cases:

- The write request occurs while the Data Cache is disabled.
- The write request is targeting a memory region marked as Normal Memory Non-Cacheable or Cacheable Write-Through.
- The write request is targeting a memory region marked as Normal Memory Cacheable Write-Back and Shareable, and the CPU is in AMP mode.

Implications

This erratum might have implications in a multi-master system where control information is passed between several processing elements in memory using a communication variable, for example a semaphore. In such a system, it is common for communication variables to be claimed using a Load-Exclusive/Store-Exclusive, but for the communication variable to be cleared using a non-Exclusive store. This erratum means that the clearing of such a communication variable might occur twice. This might lead to two masters apparently claiming a communication variable, and therefore might cause data corruption to shared data.

A scenario in which this might happen is:

```
MOV r1,#0x40; address is double-word  aligned, mapped in
                  ; Normal Non-cacheable Shareable memory
Loop:LDREXr5, [r1,#0x0]; read the communication variable
CMP r5, #0  ; check if 0
STREXEQ r5, r0, [r1]; attempt to store new value
CMPEQ r5, #0; test if store succeeded
BNE Loop; retry if not
DMB         ; ensures that all subsequent accesses are observed when
```

```
; gaining of the communication variable has been observed
; loads and stores in the critical region can now be performed
MOV r2,#0
MOV r0, #0
DMB         ; ensure all previous accesses are observed before the
; communication variable is cleared
STR r0, [r1]; clear the communication variable with normal store
STR r2, [r1,#0x4]
; previous STR might merge and be sent again, which might
; cause undesired release of the communication variable.
```

This scenario is valid when the communication variable is a byte, a half-word, or a word

Workaround

There are several possible workarounds:

- Add a DMB after clearing a communication variable:

  ```
  STR r0, [r1]; clear the communication variable
          DMB         ; ensure the previous STR is complete
  ```

  Also any IRQ or FIQ handler must execute a DMB at the start to ensure as well the clear of any communication variable is complete.

- Ensure there is no other data using the same naturally aligned 64-bit memory location as the communication variable:

  ```
                  ALIGN 64
  communication_variable DCD 0
  unused_data DCD 0
  LDR r1,= communication_variable
  ```

- Use a Store-Exclusive to clear the communication variable, rather than a non-Exclusive store.

2.4 Category B (Rare)

There are no errata in this category.

2.5 Category C

This section describes Category C errata.

2.5.1 (725631) ISB is counted in Performance Monitor events 0x0C and 0x0D

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The ISB is implemented as a branch in the Cortex-A9 micro-architecture.

This implies that events `0x0C` (software change of PC) and `0x0D` (immediate branch) are asserted when an ISB occurs, which is not compliant with the ARM Architecture.

Implications

The count of events `0x0C` and `0x0D` are not completely precise when using the Performance Monitor counters, because the ISB is counted together with the real software changes to PC (for `0x0C`) and immediate branches (`0x0D`).

The erratum also causes the corresponding PMUEVENT bits to toggle in case an ISB executes.

- PMUEVENT[13] relates to event `0x0C`
- PMUEVENT[14] relates to event `0x0D`.

Workaround

You can count ISB instructions alone with event `0x90`.

You can subtract this ISB count from the results you obtained in events `0x0C` and `0x0D`, to obtain the precise count of software change of PC (`0x0C`) and immediate branches (`0x0D`).

2.5.2 (729817) Main ID register alias addresses are not mapped on Debug APB interface

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The ARM Debug Architecture specifies registers 838 and 839 as *Alias of the Main ID register*. They should be accessible using the APB Debug interface at addresses `0xD18` and `0xD1C`.

The two alias addresses are not implemented in Cortex-A9. A read access at either of these two addresses returns 0, instead of the MIDR value.

Note

Read accesses to these two registers using the internal CP14 interface are trapped to UNDEFINED, which is compliant with the ARM Debug architecture. Therefore the erratum only applies to the alias addresses using the external Debug APB interface.

Implications

If the debugger, or any other external agent, tries to read the MIDR register using the alias addresses, it will get a faulty answer (`0x0`), which can cause indeterminate errors in the debugger afterwards.

Workaround

The workaround for this erratum is to always access the MIDR at its original address, `0xD00`, and not to use its alias address.

2.5.3 (729818) In debug state, next instruction is stalled when SDABORT flag is set, instead of being discarded

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

When the processor is in debug state, an instruction written to the ITR after a Load/Store instruction that aborts gets executed on clearing the SDABORT_l, instead of being discarded.

Conditions

- Debugger has put the extDCCmode bits into Stall mode
- A previously issued load/store instruction has generated a synchronous Data Abort (for example, an MMU fault)
- For efficiency, the debugger does not read DBGDSCRext immediately, to see if the load/store has completed and has not aborted, but writes further instructions to the ITR, expecting them to be discarded if a problem occurs
- The debugger reads the DBGDSCR at the end of the sequence and discovers the load/store aborted
- The debugger clears the SDABORT_l flag (by writing to the Clear Sticky Aborts bit in DBGDRCR).

Under these conditions, the instruction that follows in the ITR might execute instead of being discarded.

Implications

Indeterminate failures can occur because of the instruction being executed when it should not. In most cases, it is unlikely that the failure will cause any significant issue.

Workaround

There are a selection of workarounds with increasing complexity and decreasing impact. In each case the impact is a loss of performance when debugging:

1. Do not use stall mode.
2. Do not use stall mode when doing load/store operations.
3. Always check for a sticky abort after issuing a load/store operation in stall mode (the cost of this probably means workaround number 2 is a preferred alternative).
4. Always check for a sticky abort after issuing a load/store operation in stall mode before issuing any further instructions that might corrupt an important target state (such as further load/store instructions, instructions that write to live registers such as VFP, CP15).

2.5.4 (751471) DBGPCSR format is incorrect

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

With respect to the DBGPCSR register, the ARM Architecture specifies that:

- DBGPCSR[31:2] contain the sampled value of bits [31:2] of the PC.
- The sampled value is an instruction address plus an offset that depends on the processor instruction set state.
- DBGPCSR[1:0] contain the meaning of PC Sample Value, with the following permitted values:
  - `0b00` ((DBGPCSR[31:2] << 2) – 8) references an ARM state instruction
  - `0bx1` ((DBGPCSR[31:1] << 1) – 4) references a Thumb or ThumbEE state instruction
  - `0b10` IMPLEMENTATION DEFINED.

  This field encodes the processor instruction set state, so that the profiling tool can calculate the true instruction address by subtracting the appropriate offset from the value sampled in bits [31:2] of the register.

In Cortex-A9, the DBGPCSR samples the target address of executed branches (but possibly still speculative to data aborts), with the following encodings:

- DBGPCSR[31:2] contain the address of the target branch instruction, with no offset.
- DBGPCSR[1:0] contains the execution state of the target branch instruction:
  - `0b00` for an ARM state instruction
  - `0b01` for a Thumb state instruction
  - `0b10` for a Jazelle state instruction
  - `0b11` for a ThumbEE state instruction

Implications

The implication of this erratum is that the debugger tools must not rely on the architected description for the value of DBGPCSR[1:0], nor remove any offset from DBGPCSR[31:2], to obtain the expected PC value.

Subtracting 4 or 8 from the DBGPCSR[31:2] value would lead to an area of code which is unlikely to have been recently executed, or which might not contain any executable code.

The same might be true for Thumb instructions at half-word boundaries, in which case PC[1]=1 but DBGPCSR[1]=0, or ThumbEE instructions at word boundaries, with PC[1]=0 and DBGPCSR[1]=1.

In Cortex-A9, because the DBGPCSR is always a branch target (= start of a basic block to the tool), the debugger should be able to spot many of these cases and attribute the sample to the right basic block.

Workaround

The debugger tools can find the expected PC value and instruction state by reading the DBGPCSR register, and consider it as described in the Description section.

2.5.5 (752519) An imprecise abort might be reported twice on non-cacheable reads

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r2, r3 and r4 revisions. Open

Description

In the case when two outstanding read memory requests to device or non-cacheable normal memory regions are issued by the Cortex-A9, and the first one receives an imprecise external abort, then the second access might falsely report an imprecise external abort.

Conditions

The erratum can only happen in systems which can generate imprecise external aborts on device or non-cacheable normal memory regions accesses.

Implications

When the erratum occurs, a second, spurious imprecise abort might be reported to the core when it should not.

In practice, the failure is unlikely to cause any significant issues to the system because imprecise aborts are usually unrecoverable failures. Because the spurious abort can only happen following a first imprecise abort, either the first abort is ignored - and the spurious abort is then ignored too -, or it is acknowledged and probably generates a critical failure in the system, such as a processor reset or whole system reboot.

Workaround

There is no practical software workaround for the erratum.

2.5.6 (754323) Repeated Store in the same cache line might delay the visibility of the Store

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All r2, r3 and r4 revisions. Open

Description

Since r2p0 revision, the Cortex-A9 implements a small counter which ensures the external visibility of all stores in a finite amount of time, causing an eventual drain of the Merging Store Buffer. This is to avoid erratum #754327, where written data could potentially remain indefinitely in the Store Buffer.

This Store Buffer has merging capabilities, and continues to merge data as long as the write accesses are performed in the same cache line. The issue which causes this erratum is that the draining counter resets each time a new data merge is performed.

In the case when a code sequence loops, and continues to write data in this same cache line, then the external visibility of the written data might not be ensured.

A livelock situation might consequently occur if any external agent is relying on the visibility of the written data, and where the writing processor cannot be interrupted while doing its writing loop.

Conditions

The erratum can only happen on Normal Memory regions.

The following examples describe scenarios that might trigger the erratum:

1. The processor continues incrementing a counter, writing the same word at the same address. The external agent (possibly another processor) polls on this address, waiting for any update of the counter value to proceed.

   The Store Buffer continues merging the updated value of the counter in its cache line, so that the external agent never sees any updated value, possibly leading to livelock.

2. The processor writes a value in a given word to indicate completion of its task, then continues writing data in an adjacent word in the same cache line.

   The external agent continues to poll the first word memory location to check when the processor completes its task. The situation is the same in the first example, because the cache line might remain indefinitely in the merging Store Buffer, creating a possible livelock in the system.

Implications

This erratum might create performance issues, or a worst case livelock scenario, if the external agent relies on the automatic visibility of the written data in a finite amount of time.

Workaround

The recommended workaround for this erratum is to insert a DMB operation after the faulty write operation in code sequences that this erratum might affect, to ensure the visibility of the written data to any external agent.

2.5.7 (756421) Sticky Pipeline Advance bit cannot be cleared from debug APB accesses

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The Sticky Pipeline Advance bit is bit[25] of the DBGDSCR register. This bit enables the debugger to detect whether the processor is idle. This bit is set to 1 every time the processor pipeline retires one instruction.

A write to DBGDRCR[3] clears this bit.

The erratum is that the Cortex-A9 does not implement any debug APB access to DBGDRCR[3].

Implications

Because of the erratum, the external debugger cannot clear the Sticky Pipeline Advance bit in the DBGDSCR. In practice, this makes the Sticky Pipeline Advance bit concept unusable on Cortex-A9 processors.

Workaround

There is no practical workaround for this erratum.

The only possible way to reset the Sticky Pipeline Advance bit is to assert the nDBGRESET input pin on the processor, which obviously has the side effect to reset all debug resources in the concerned processor, and any other additional Coresight components nDBGRESET connects to.

2.5.8 (757119) Some *Unallocated memory hint* instructions generate an Undefined Instruction exception instead of being treated as NOP

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The ARM Architecture specifies that ARM opcodes of the form `11110 100x001 xxxx xxxx xxxx xxxx xxxx` are *Unallocated memory hint (treat as NOP)* if the core supports the MP extensions, as the Cortex-A9 does.

The errata is that the Cortex-A9 generates an Undefined Instruction exception when bits [15:12] of the instruction encoding are different from `4'b1111`, instead of treating the instruction as a NOP.

Implications

Because of the erratum, an unexpected Undefined Instruction exception might be generated.

In practice, this erratum is unlikely to cause any significant issue because such instruction encodings are not supposed to be generated by any compiler, nor used by any handcrafted program.

Workaround

The workaround for this erratum is to modify the instruction encoding with bits[15:12]=`4'b1111`, so that the Cortex-A9 treats the instruction properly as a NOP.

If it is not possible to modify the instruction encoding as described, the Undefined Instruction exception handler has to cope with this case, and emulate the expected behavior of the instruction, that is, it must do nothing (NOP), before returning to normal program execution.

2.5.9 (761321) MRC and MCR are not counted in event 0x68

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

Event `0x68` counts the total number of instructions passing through the register rename pipeline stage. The erratum is that MRC and MCR instructions are not counted in this event.

The event is also reported externally on PMUEVENT[9:8], which suffers from the same defect.

Implications

The implication of this erratum is that the values of event `0x68` and PMUEVENT[9:8] are imprecise, omitting the number of MCR and MRC instructions. The inaccuracy of the total count depends on the rate of MRC and MCR instructions in the code.

Workaround

No workaround is possible to achieve the required functionality of counting precisely how many instructions are passing through the register rename pipeline stage when the code contains some MRC or MCR instructions.

2.5.10 (764319) Read accesses to DBGPRSR and DBGOSLSR may generate an unexpected Undefined Instruction exception

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

CP14 read accesses to the DBGPRSR and DBGOSLSR registers generate an unexpected Undefined Instruction exception when the DBGSWENABLE external pin is set to 0, even when the CP14 accesses are performed from a privileged mode.

Implications

Because of the erratum, the DBGPRSR and DBGOSLSR registers are not accessible when DBGSWENABLE=0.

This is unlikely to cause any significant issue in Cortex-A9 based systems because these accesses are mainly intended to be used as part of debug over powerdown sequences, and the Cortex-A9 does not support this feature.

Workaround

The workaround for this erratum is to temporarily set the DBGSWENABLE bit to 1 so that the DBGPRSR and DBGOSLSR registers can be accessed as expected.

There is no other workaround for this erratum.

2.5.11 (771224) Visibility of Debug Enable access rights to enable/disable tracing is not ensured by an ISB

Status

Affects: Product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

According to the ARM architecture, any change in the Authentication Status Register should be made visible to the processor after an exception entry or return, or an ISB.

Although this is correctly achieved for all debug-related features, the ISB is not sufficient to make the changes visible to the trace flow. As a consequence, the **WPTTRACEPROHIBITEDn** signal(s) remain stuck to their old value up to the next exception entry or return, or to the next serial branch, even when an ISB executes.

A serial branch is one of the following:

- Data processing to PC with the S bit set (for example, `MOVS pc, r14`)
- `LDM pc ^`

Implications

Because of the erratum, the trace flow might not start, nor stop, as expected by the program.

Workaround

To work around the erratum, the ISB must be replaced by one of the events causing the change to be visible. In particular, replacing the ISB by a MOVS PC to the next instruction will achieve the correct functionality.

2.5.12 (775419) PMU event 0x0A (exception return) might count twice the LDM PC ^ instructions with base address register write-back

Status

Affects: product Cortex-A9, Cortex-A9 MPCore.

Fault Type: Programmer Category C

Fault Status: Present in: All revisions. Open

Description

The LDM PC ^ instructions with base address register write-back might be counted twice in the PMU event 0x0A, which is counting the number of exception returns.

The associated PMUEVENT[11] signal is also affected by this erratum, and might be asserted twice by a single LDM PC ^ with base address register write-back.

Implications

Because of the erratum, the count of exception returns is imprecise. The error rate depends on the ratio between exception returns of the form LDM PC ^ with base address register write-back and the total number of exceptions returns.

Workaround

There is no workaround to this erratum.